小马哥的 Java 项目实战营

Java EE 项目 - 第七节 Maven 项目管理

小马哥（mercyblitz）

我是谁?

小马哥（mercyblitz）

- 父亲
- Java 劝退师
- Apache Dubbo PMC
- Spring Cloud Alibaba 架构师
- 《Spring Boot 编程思想》作者

议题

- Maven 基础
- Maven 生命周期
- Maven 常用插件
- Maven 插件扩展
- 问答互动

Maven 基础

- 什么是 Maven？

  Maven是一个意思是知识积累者的Yiddish词，开始试图简化 Jakarta Turbine 项目的构建过程。有几个项目，每个项目都有自己的 Ant 构建文件，这些都略有不同。罐子被检查成CVS。我们想要一种标准的方式来构建项目，明确定义项目组成内容，发布项目信息的简便方法，以及在多个项目中共享JAR的方法。 Maven 现在可以用于构建和管理任何基于Java的项目。我们希望我们已经创建了一些能够使Java开发人员的日常工作更容易，并且通常有助于理解任何基于Java的项目。

Maven 基础

- Maven 的目标

Maven的主要目标是让开发人员在最短的时间内理解开发工作的完整状态。为了实现这一目标，Maven处理了几个值得关注的领域：

- 使构建过程变得简单
- 提供统一的构建系统
- 提供高质量的项目信息
- 鼓励更好的发展实践

Maven 基础

- 使构建过程变得简单

  虽然使用Maven并没有消除了解底层机制的需要，但Maven确实为开发人员提供了许多细节。

- 提供统一的构建系统

  Maven使用其项目对象模型（POM）和一组插件构建项目。一旦你熟悉了一个Maven项目，你就知道所有Maven项目都是如何构建的。这可以在导航许多项目时节省时间。

Maven 基础

- 提供高质量的项目信息

  Maven提供有用的项目信息，部分来自POM，部分来自项目来源。例如，Maven可以提供

  - 直接从源码控制创建更改日志
  - 交叉引用的来源
  - 由项目管理的邮件列表
  - 项目使用的依赖关系
  - 单位测试报告，包括覆盖范围

Maven 基础

- 鼓励更好的发展实践

Maven旨在收集最佳实践开发的当前原则，并轻松指导项目朝这个方向发展。例如，单元测试的规范，执行和报告是使用Maven的正常构建周期的一部分。目前的单位测试最佳实践被用作指导方针：

- 将测试源代码保存在单独但并行的源树中
- 使用测试用例命名约定来定位和执行测试
- 让测试用例设置他们的环境，而不是自定义构建以进行测试准备

Maven 基础

- 功能列表
  - 遵循最佳实践的简单项目设置-在几秒钟内启动新项目或模块
  - 所有项目的一致使用-意味着新开发人员没有时间进入项目
  - 高级依赖管理，包括自动更新，依赖闭包（也称为传递依赖关系）
  - 能够轻松地同时处理多个项目
  - 一个庞大且不断增长的图书馆和元数据存储库，用于开箱即用，并与最大的开源项目进行安排，以便实时提供最新版本
  - 可扩展，能够轻松地用Java或脚本语言编写插件
  - 即时访问几乎没有或没有额外配置的新功能
  - Ant 任务的依赖管理和部署在Maven之外

Maven 基础

- 功能列表
  - 基于模型的构建：Maven能够基于项目的元数据将任意数量的项目构建到预定义的输出类型中，例如JAR，WAR或分发，而在大多数情况下无需执行任何脚本
  - 项目信息的一致站点：使用与构建过程相同的元数据，Maven能够生成网站或PDF，包括您要添加的任何文档，并添加到该标准报告中有关项目开发状态的报告。此信息的示例可以在“项目信息”和“项目报告”子菜单下的该站点左侧导航的底部看到
  - 发布管理和分发发布：如果没有太多额外的配置，Maven将与您的源码控制系统（如SVN或Git）集成，并基于某个标签管理项目的发布。它还可以将其发布到分发位置以供其他项目使用。Maven能够发布单个输出，例如JAR，包含其他依赖项和文档的存档，或作为源码分发

Maven 基础

- 功能列表
  - 依赖管理：Maven鼓励使用JAR和其他依赖项的中央存储库。Maven提供了一种机制，您的项目客户端可以使用该机制从中央JAR存储库下载构建项目所需的任何JAR，就像Perl的CPAN一样。这允许Maven的用户在项目之间重用

Maven 基础

- 什么是 POM？

项目对象模型（POM）是Maven的基本工作单元。它是一个XML文件，包含有关项目的信息和Maven用于构建项目的配置详细信息。它包含大多数项目的默认值。例如build目录，它是target；source目录，它是src/main/java；test source目录，它是src/test/java；等等。当执行任务或目标时，Maven在当前目录中查找POM。它读取POM，获取所需的配置信息，然后执行目标。POM中可以指定的一些配置包括项目依赖项、可以执行的插件或目标、构建概要文件等等。还可以指定其他信息，如项目版本、描述、开发人员、邮件列表等。

Maven 基础

- 项目继承

  POM中合并的元素如下：

  - dependencies
  - developers and contributors
  - plugin lists (including reports)
  - plugin executions with matching ids
  - plugin configuration
  - resources

Maven 基础

- 项目聚合

项目聚合类似于项目继承。但是它不是从模块中指定父POM，而是从父POM中指定模块。通过这样做，父项目现在知道了它的模块，如果对父项目调用了Maven命令，那么Maven命令也将被执行到父项目的模块中。要进行项目聚合，必须执行以下操作：

- 将父POMs packaging更改为值“pom”。
- 在父POM中指定其模块（子POM）的目录。

Maven 基础

- 项目继承 V.S 项目聚合

  如果您有几个Maven项目，并且它们都具有相似的配置，那么您可以通过提取这些相似的配置并生成父项目来重构您的项目。因此，您所要做的就是让Maven项目继承父项目，然后这些配置将应用于所有这些项目。

  如果您有一组一起生成或处理的项目，则可以创建一个父项目，并让该父项目将这些项目声明为其模块。这样做，您只需构建父级，其余的就可以了。

Maven 基础

- 项目继承 V.S 项目聚合

  当然，您可以同时拥有项目继承和项目聚合。也就是说，您可以让您的模块指定父项目，同时让父项目指定那些Maven项目作为其模块。你只需要应用这三条规则：

  - 在每个子POM中指定其父POM是谁。
  - 将父POMs packaging更改为值“pom”。
  - 在父POM中指定其模块（子POM）的目录

Maven 基础

- 项目插值和变量

Maven鼓励的一个做法是不要重复你自己。但是，在某些情况下，您需要在多个不同的位置使用相同的值。为了帮助确保值只指定一次，Maven允许您在POM中使用自己的和预定义的变量。

需要注意的一个因素是，如上所述，这些变量是在继承之后处理的。这意味着，如果父项目使用变量，那么最终使用的将是它在子项目中的定义，而不是父项目中的定义。

Maven 基础

- 可用变量

  作为单个值元素的模型的任何字段都可以作为变量引用。例如${项目.groupId}, ${项目.版本}, ${project.build.source目录}等等。请参阅POM参考以查看完整的属性列表。

  这些变量都由前缀“project”引用。你也可以看到pom的引用。作为前缀，或完全省略前缀-这些形式现在已被弃用，不应使用。

Maven 基础

- 构建配置（Build Profiles）

Apache Maven竭尽全力确保构建是可移植的。除此之外，这意味着允许在POM内进行构建配置，避免所有文件系统引用（在继承、依赖项和其他位置），并且更加依赖于本地存储库来存储实现这一点所需的元数据。

然而，有时可移植性并不是完全可能的。在某些情况下，插件可能需要配置本地文件系统路径。在其他情况下，需要稍微不同的依赖集，并且项目的工件名称可能需要稍微调整。还有一些时候，甚至可能需要在构建生命周期中包含一个完整的插件，具体取决于检测到的构建环境。

Maven 基础

- 构建配置（Build Profiles）

为了解决这些情况，Maven支持构建概要文件。概要文件是使用POM本身中可用的元素子集（加上一个额外的部分）指定的，并以各种方式触发。它们在构建时修改POM，并用于互补集合中，为一组目标环境提供等效但不同的参数（例如，提供开发、测试和生产环境中appserver根的路径）。因此，概要文件很容易导致团队中不同成员的不同构建结果。但是，如果使用得当，可以在保持项目可移植性的同时使用概要文件。这也将最小化maven的-f选项的使用，该选项允许用户创建另一个具有不同参数或配置的POM来构建，这使得它更易于维护，因为它只使用一个POM运行。

Maven 基础

- 不同类型的配置（Profiles）
  - 单个工程

  在POM本身中定义(pom.xml文件)

  - 单个用户

  在Maven设置（%USER\u HOME%/.m2）中定义/设置.xml)

  - 全局

  Defined in the global Maven-settings (${maven.home}/conf/settings.xml)

Maven 基础

- 激活配置（Profiles）

  配置文件可以通过多种方式激活：

  - 从命令行
  - 通过Maven设置
  - 基于环境变量
  - 操作系统设置
  - 存在或缺少文件

Maven 基础

- 标准目录结构

拥有一个公共目录结构，让熟悉一个Maven项目的用户能够在另一个Maven项目中立即感到宾至如归。其优点类似于采用全站式外观和感觉。

- src/main/java – 应用程序/库源代码
- src/main/resources – 应用程序/库资源
- src/main/webapp – Web应用程序源
- src/test/java – 测试源代码
- src/test/resources – 测试资源
- src/it – 集成测试（主要针对插件）
- src/assembly – 程序集描述符

Maven 基础

- 标准目录结构
  - src/site – 站点
  - LICENSE.txt – 项目许可证
  - NOTICE.txt – 项目所依赖的库所需的通知和属性
  - README.txt – 项目自述

Maven 基础

- 依赖机制

依赖关系管理是Maven的一个核心特性。管理单个项目的依赖关系很容易。管理由数百个模块组成的多模块项目和应用程序的依赖关系是可能的。Maven在定义、创建和维护具有良好定义的类路径和库版本的可复制构建方面提供了很大的帮助。

Maven 基础

- 可传递依赖

Maven通过自动包含可传递的依赖项，避免了发现和指定您自己的依赖项所需的库的需要。

通过从指定的远程存储库读取依赖项的项目文件，可以简化此功能。通常，这些项目的所有依赖项都会在您的项目中使用，就像项目从其父项目或其依赖项继承的任何依赖项一样，依此类推。

可以从中收集依赖项的级别数量没有限制。只有在发现循环依赖时才会出现问题。

有了可传递的依赖关系，包含的库的图可以很快变大。因此，还有一些附加功能限制了包含哪些依赖项：

Maven 基础

- 可传递依赖
  - 依赖仲裁（Dependency mediation）- 这决定了当多个版本作为依赖项遇到时，将选择工件的哪个版本。Maven选择“最接近的定义”。也就是说，它使用依赖关系树中与项目最接近的依赖关系的版本。您可以通过在项目的POM中显式声明来保证一个版本。请注意，如果两个依赖关系版本在依赖关系树中处于同一深度，则第一个声明获胜
  - 依赖管理（Dependency management）- 这允许项目作者直接指定在可传递依赖项或未指定版本的依赖项中遇到的工件的版本。在上一节中的示例中，依赖项直接添加到了，即使a没有直接使用它。相反，a可以将D作为依赖项包含在其dependencyManagement节中，并直接控制在引用D时使用哪个版本的D

Maven 基础

- 可传递依赖
  - 依赖作用域（Dependency scope）- 这允许您只包含适用于构建的当前阶段的依赖项。下面将对此进行更详细的描述
  - 排除依赖（Excluded dependencies）- 如果项目X依赖于项目Y，而项目Y依赖于项目Z，那么项目X的所有者可以使用“exclusion”元素显式排除项目Z作为依赖项
  - 可选依赖（Optional dependencies） - 如果项目Y依赖于项目Z，那么项目Y的所有者可以使用“optional”元素将项目Z标记为可选依赖项。当项目X依赖于项目Y时，X将只依赖于Y而不依赖于Y的可选依赖项Z。然后，项目X的所有者可以根据自己的选择显式地添加对Z的依赖项。（将可选依赖项视为“默认排除”可能会有所帮助。）

Maven 基础

- 依赖作用域

依赖作用域用于限制依赖项的可传递性，并确定依赖项何时包含在类路径中。共有6个作用域：

- compile - 如果没有指定，则使用默认范围。编译依赖项在项目的所有类路径中都可用。此外，这些依赖关系被传播到依赖项目

- provided - 这很像compile，但表示您希望JDK或容器在运行时提供依赖关系。例如，在为javaenterpriseedition构建web应用程序时，您可以将对Servlet API和相关javaeeapi的依赖关系设置为提供的范围，因为web容器提供了这些类。具有此作用域的依赖项将添加到用于编译和测试的类路径，而不是运行时类路径。它是不可传递的

Maven 基础

- 依赖作用域
  - runtime - 此范围表示编译时不需要依赖项，但执行时需要依赖项。Maven在运行时和测试类路径中包含具有此范围的依赖项，但在编译类路径中不包含

  - test - 此范围表示应用程序的正常使用不需要依赖关系，仅在测试编译和执行阶段可用。此范围不可传递。通常这个范围用于测试库，比如JUnit和Mockito。如果非测试库（如apachecommons IO）用于单元测试（src/test/java），而不用于模型代码（src/main/java），那么它也可以用于这些库

Maven 基础

- 依赖作用域
  - system - 这个范围与provided类似，只是您必须提供显式包含它的JAR。工件总是可用的，不会在存储库中查找
  - import - 只有<dependencyManagement>部分中pom类型的依赖项才支持此范围。它表示将用指定POM的<dependencyManagement>部分中的有效依赖项列表替换依赖项。因为它们被替换了，所以具有导入范围的依赖项实际上并不参与限制依赖项的可传递性

Maven 基础

- 依赖作用域

每个作用域（ import 除外）都以不同的方式影响可传递依赖项，如下表所示。如果将依赖项设置为左列中的范围，则该依赖项的可传递依赖项（该依赖项的范围在顶行中）将导致主项目中的依赖项（该依赖项的范围在相交处列出）。如果没有列出任何范围，则表示忽略了依赖关系

	compile	provided	runtime	test
compile	compile(*)	-	runtime	-
provided	provided	-	provided	-
runtime	runtime	-	runtime	-
test	test	-	test	-

Maven 基础

- 依赖管理

  依赖关系管理部分是一种集中依赖关系信息的机制。当您有一组从公共父级继承的项目时，可以将有关依赖关系的所有信息放在公共POM中，并对子POM中的工件进行更简单的引用

Maven 基础

- 可选依赖

当无法（无论出于何种原因）将项目拆分为子模块时，使用可选依赖项。其思想是，某些依赖项仅用于项目中的某些特性，如果不使用该特性，则不需要这些依赖项。理想情况下，这样一个特性将被分解成一个子模块，该子模块依赖于核心功能项目。这个新的子项目只有非可选依赖项，因为如果您决定使用子项目的功能，那么您将需要它们。

但是，由于项目不能拆分（无论出于何种原因，再次），这些依赖项被声明为可选的。如果用户想要使用与可选依赖项相关的功能，他们必须在自己的项目中重新声明该可选依赖项。这不是处理这种情况的最清晰的方法，但是可选的依赖项和依赖项排除都是权宜之计。

Maven 基础

- 依赖排除

  由于Maven以传递方式解析依赖项，因此不需要的依赖项可能会包含在项目的类路径中。例如，某个较旧的jar可能存在安全问题，或者与您正在使用的Java版本不兼容。为了解决这个问题，Maven允许您排除特定的依赖关系。排除是在POM中的特定依赖项上设置的，并且针对特定的groupId和artifactId。在构建项目时，不会通过声明排除的依赖项将该工件添加到项目的类路径中。

Maven 生命周期

- 构建生命周期

Maven基于构建生命周期的核心概念。这意味着构建和分发特定工件（项目）的过程是明确定义的。

对于构建项目的人来说，这意味着只需要学习一小部分命令就可以构建任何Maven项目，POM将确保他们获得所需的结果。

有三个内置的构建生命周期：default、clean和site。默认生命周期处理项目部署，清理生命周期处理项目清理，而站点生命周期处理项目站点文档的创建。

Maven 生命周期

- 生命周期阶段

  每个构建生命周期都由不同的构建阶段列表定义，其中构建阶段表示生命周期中的一个阶段。例如，默认生命周期由以下阶段组成：

  - validate - 验证项目是否正确，所有必要的信息是否可用
  - compile - 编译项目的源代码
  - test - 使用合适的单元测试框架测试编译的源代码。这些测试不应该要求打包或部署代码

Maven 生命周期

- 生命周期阶段
  - package – 把编译好的代码打包成可分发的格式，比如JAR
  - verify – 对集成测试的结果进行检查，以确保符合质量标准
  - install – 将包安装到本地存储库中，作为本地其他项目的依赖项使用
  - deploy – 在构建环境中完成，将最终包复制到远程存储库，以便与其他开发人员和项目共享

Maven 常用插件

- 常用 Maven 插件
  - compiler
  - source
  - javadoc
  - deploy
  - shade
  - flatten
  - surefire

THANKS!